Disambiguation Tools for NooJ

Max Silberztein

Disambiguation Tools for NooJ

Max Silberztein
LASELDI, Université de Franche-Comté
max.silberztein@gmail.com

Abstract: When NooJ performs an automatic lexical analysis of corpora, it recognizes five types of atomic linguistic units (ALUs) and represents them as annotations stored inside each text's annotation structure (TAS). Unfortunately, the massive level of ambiguities generated by each of the five corresponding parsers produces a TAS far too heavy for most corpus linguistics applications. In consequence, most users' queries produce too many incorrect results. In order to provide a working solution for NooJ's lexical parser's behavior, we have implemented a new set of tools specifically designed to deal with unwanted ambiguities in corpora and texts: automatic and semi-automatic tools as well as a manual access to edit the TAS.

Introduction

With NooJ, linguists can represent five types of Atomic Linguistic Units (ALUs)[1]:

- Affixes, morphemes and components of contracted words, e.g. *dis-*, *-ization*, *cannot*
- simple words and their morphological variants, e.g. *a laugh, to laugh, laughed, laughable*
- multi-word units, semi-frozen terms and their variants, e.g. *as a matter of fact, a nuclear submarine*
- local syntactic units, e.g. complex determiners, dates, e.g. *Most of my groups of, Monday June 5th in the early afternoon*
- discontinuous expressions such as collocations, support verb constructions, phrasal verbs, e.g. *to take ... into account, to give ... in*

Accordingly, NooJ provides tools to describe these five types of ALUs and recognize them in corpora automatically: dictionaries, morphological Finite State Transducers (FSTs), Recursive Transition Networks (RTNs), as well as the ability to link dictionaries and syntactic grammars to formalize lexicon-grammar tables[2].

Each level of analysis constitutes an autonomous module. NooJ processes each module one after the other, in cascade form. Therefore, each module must produce a result that has 100% recall, so that the subsequent module processes an input which lacks no potential linguistic hypothesis. The cost of this approach is that the accuracy of each module is very low: all potential linguistic hypotheses have to be produced and transmitted to the next module, however improbable they might be.

The accuracy of a lexical parser will get lower and lower as the precision of the linguistic data increases: for instance, a simple parser might process the word form *will* as two-time ambiguous (Noun or Verb) whereas a more sophisticated parser will distinguish the different meanings of the noun *will* (a mental faculty, a legal declaration) as well as of the verb (used to introduce the future, or synonymous to *to wish*). Clearly, the more sophisticated lexical parser produces a higher level of ambiguity than the simple one, and it takes a more sophisticated linguistic analysis to solve the extra ambiguities.

An ideally tagged text

Because any linguistic analysis must process all types of ALUs (not only the simple words), an ideal tagger should produce a result that looks like the following:

Battle-tested/A Japanese/A industrial managers/N here/ADV always/ADV buck up/V nervous/A newcomers/N with/PREP the/DET tale/N of/PREP the first of their/N countrymen/N to/PREP visit/V

[1] See the NooJ manual, which is updated regularly (Silberztein 2002).
[2] See (Vietri 2008) for examples of lexicon-grammar tables and their formalization in NooJ.

Mexico/LOC, a boatload of/DET samurai warriors/N blown ashore/VPP 375 years ago/DATE. From the beginning/DATE, it took/EXP1 a/DET man/N with/PREP extraordinary/A qualities/N to/EXP1 succeed/V in/PREP Mexico/LOC, says/V Kimihide Takimura/NPR, president/N of/PREP Mitsui/NPR group's/N Kensetsu Engineering Inc./ORG unit/N.

For instance, it is crucial to tag the multi-word noun *industrial managers* as a unit, as opposed to produce the two tags "industrial/A" and "managers/N". In most cases, the semantic analysis of the sequence "industrial <N>" is:

Industrial <N> = <N> produced by industrial methods

This productive analysis can successfully be applied to a large number of nouns, such as in the following examples:

Industrial cheese = cheese produced by industrial methods
Industrial food = food produced by industrial methods
Industrial cloth = cloth produced by industrial methods
etc.

But this productive analysis does not apply to *industrial manager*. If one wants to translate this sequence correctly in French, one has to translate it as a whole, and not word by word. For instance, a correct French translation would be "patron de PME", whereas "gestionnaire industriel" sounds odd at the very least, and is never used.

If it was possible to automatically tag texts *correctly*, i.e. to automatically annotate all types of linguistic units (and not only simple words), and to produce a 100% correct result, then the example above would constitute a good way to represent the result of the lexical analysis of texts.

Unfortunately, any lexical parser that aims to represent all types of linguistic units while producing a result with 100% recall will produce a high degree of ambiguities, because it is not always possible to remove all ambiguities at the lexical level. For instance, consider the following text:

… There is a round table in room A32 …

The only way an automatic parser could (maybe!) reliably choose between the analysis "round table = meeting" and the analysis "round table = round piece of furniture" is to perform some complex discourse analysis that would take a larger context into account. It is impossible to choose between these two solutions at a lexical level, i.e. at the level taggers and lexical parsers operate. Taggers designed to remove ambiguities at any cost, even when it is impossible to do so reliably, simply ignore multi-word units and use probability or other techniques-based heuristics to "flip coins" and therefore produce results that are useless for many precise NLP applications. Ambiguities are generated at each level of the five lexical analyses. For instance:

— the morphological parser analyzes the word form *recollect* as the prefix *re* followed by the verb *collect* (meaning: collect again), whereas the dictionary lookup will analyze it as a simple verb (meaning: to control oneself). It would take a sophisticated semantic analysis (at least) to choose between the two solutions in the following text:

John recollects the old coins

Does he remembers about them, or did he decide to take his collecting hobby back up again?

— the multi-word recognizer analyses the sequence *acid rock* as a noun (a style of rock music), whereas the simple word recognizer analyses it as a sequence of an adjective followed by a noun.

— there are several phrasal verb entries for "to back up", depending on the distributional property of their object complements. How could a lexical parser choose between these entries without a distributional

analysis of the complement, and without a reference analysis when the complement is implicit? For instance:

John backed his car up → John backed it up (he drove in reverse)
John backed Mary's statement up → John backed it up (he supported her statement)
John backed Mary's idea up → John backed it up (he proved her idea right)
John backed his computer up → John backed it up (he saved its files)

In conclusion: it is impossible to build an automatic lexical parser that would disambiguate all the linguistic units that occur in texts. Automatic taggers, which aim at this very goal, simply cannot be relied upon. The only way to build a *reliable* lexical parser is to allow it to represent unsolvable lexical ambiguities. These ambiguities will have to be passed to subsequent parsers that would use syntactic, semantic and/or discourse analysis techniques to solve them (and not in all cases, because there are ambiguous sentences in texts!).

NooJ's Text Annotation Structure (TAS)

NooJ's lexical parser uses parallel annotations, rather than linear tags, to represent the lexical analyses of texts[3]. Annotations have two advantages over tags:

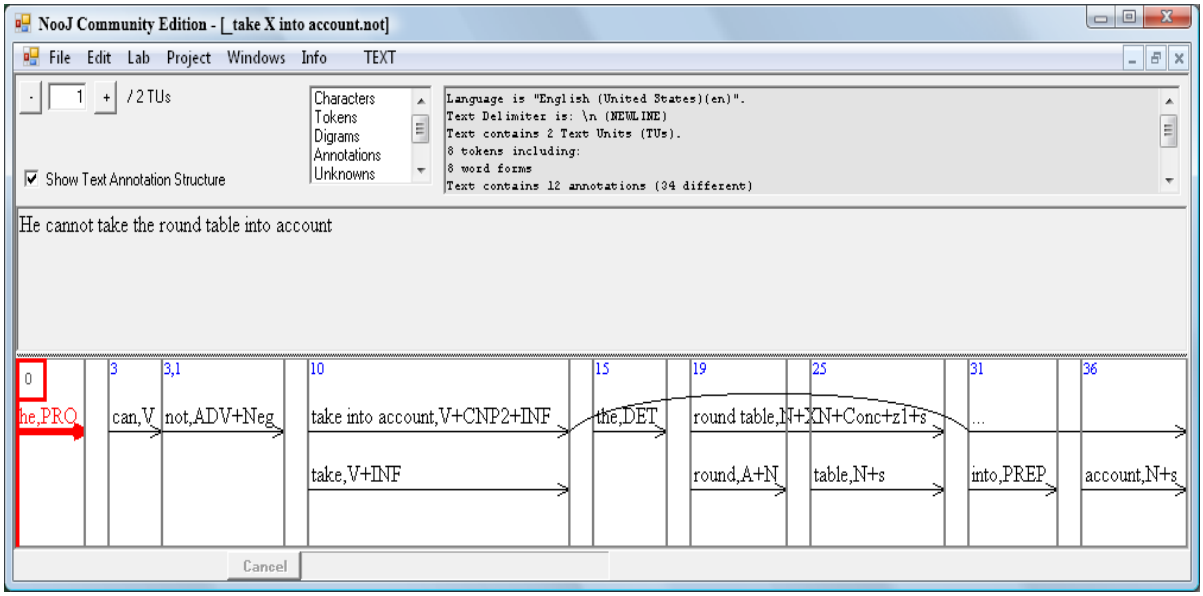
— they can represent all types of linguistic units, such as affixes (inside word forms), multi-word units and discontinuous linguistic units[4];

— they can be stacked together and therefore represent lexical ambiguities.

For instance, consider the following sentence:

He cannot take the round table into account

In this sentence, the word form *cannot* corresponds to a sequence of two linguistic units; the sequence *round table* is ambiguous, and the linguistic unit *take into account* is discontinuous.



The TAS's main application is that no potential linguistic unit is left out: for instance, if the next sentence in the text is: "we will have to postpone it", a semantic parser can infer that round table refers to a meeting, not to a

[3] See (Silberztein 2006) for a description of NooJ's Annotation engine, and (Silberztein 2007) for linguistic applications of the Text Annotation Structure.
[4] See in particular how discontinuous expressions are represented in the Text Annotation Structure in (Silberztein 2008).

piece of furniture. Conversely, if the next sentence in the text is: “we will have to move it closer to the window”, the semantic parser can infer that the round table is a piece of furniture. Both solutions are alive and ready to be chosen.

Another advantage of the TAS is that it unifies all types of linguistic units. From now on, any subsequent parser will process annotations rather than affixes, simple or multi-word units and discontinuous expressions. For instance, the following NooJ query:

```
<V> <ADV> <V>
```

which stands for: extract from the corpus all sequences of a verb, followed by an adverb, followed by a verb, will produce concordances in which various types of sequences are displayed:

... John has often taken ...
... He cannot take into account the ...
... She is not finished with ...

This characteristic gives NooJ a higher recall factor than other corpus linguistic tools, and is absolutely crucial when dealing with agglutinated languages. For instance, in Arabic, “and in my house” is written as a single word, but NooJ will still process it as a sequence of four distinct annotations. Beyond the simple corpus linguistics applications, the fact that the lexical parser deals with complex word forms allows the subsequent syntactic grammars to be much easier to read and write. For instance, when linguists formalize Romance Languages’ noun phrases, they need to take pay special attention to the determiner, which often is contracted with the preceding preposition or adverb. For instance, in French, the word form *du* is either a determiner or the contraction of the preposition *de* followed by the determiner *le*. In practice, this simple problem often leads to an unacceptable number of duplicate and redundant rules in the formalization of noun phrases.

With NooJ however, linguists just have to describe the head of noun phrases independently from the contraction problem, which is solved beforehand by the lexical parser.

Disambiguated texts are useful

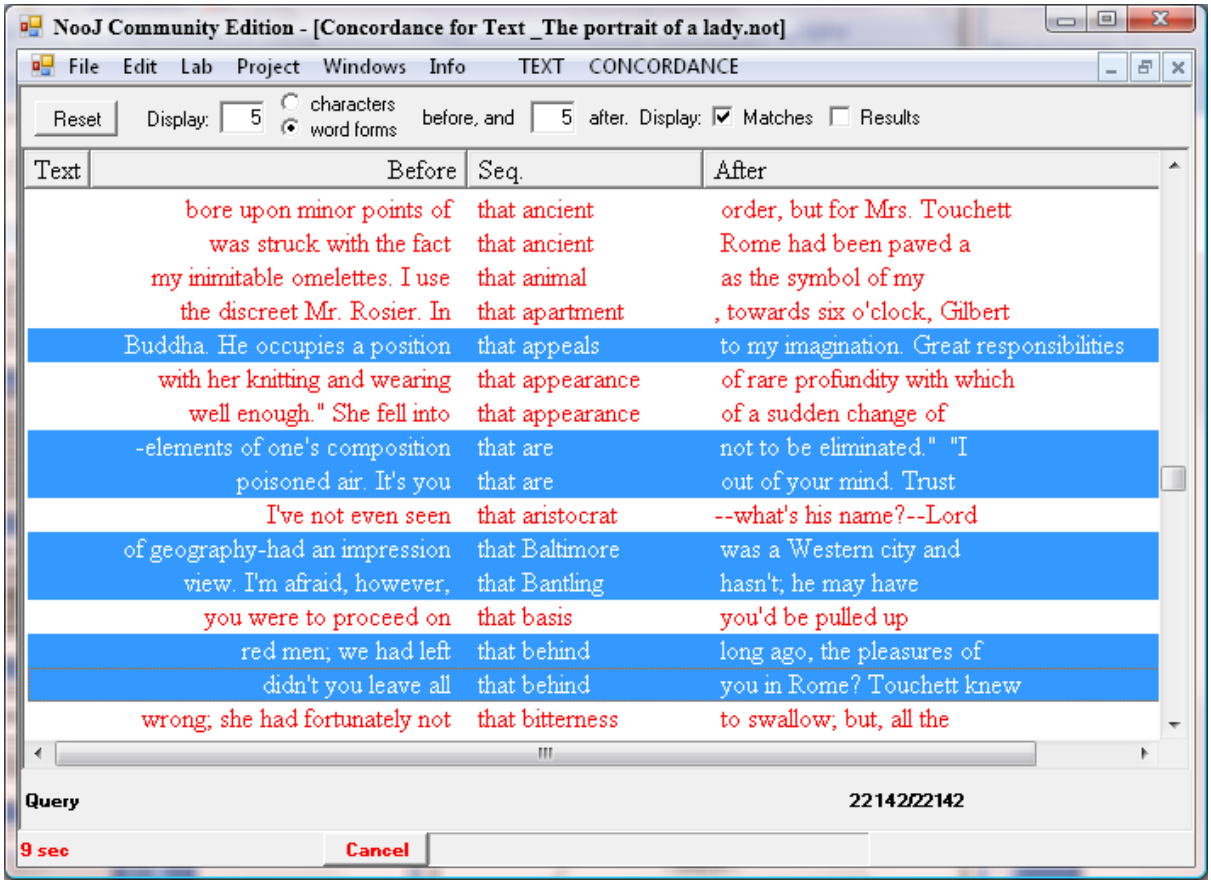
Unfortunately, using fine-grained lexical resources with a TAS in which all ambiguities are retained makes NooJ’s corpus processing system produce noisy results that are very unattractive to linguists who are used to tagger-based corpus processors that produce much less noise, at the expense of some silence (but of course users don’t see the silence).

A more serious problem is related to the use of NooJ as a corpus processing system, especially by non-linguists who want to analyze texts in other applications: literary, psychological, sociological analyses, etc.

Leaving a large number of ambiguities in TAS often produce very *noisy* results when a user applies a simple query. For instance, the simple query:

```
<DET> <N>
```

(look for all determiners followed by a noun), when applied to the text The *portrait of a lady* (Henry James, 1881) produces a 22,142-entry concordance with almost 50% noise:



(incorrect matches are selected in the figure). This result is arguably improvable! Let's examine the incorrect results:

... that appeals ...

This sequence has wrongly been brought up because *that* could be a determiner, and *appeals* could be a noun. That is not the case though: if *that* were a determiner, we know that it would be a singular one because it is described as such in the English dictionary. However, if the word form *appeals* were a noun, it would be in the plural. The sequence "that/DET appeals/N" violates the agreement rule between determiners and the following noun; therefore this sequence cannot correspond to the query <DET> <N>. We can remove a large number of incorrect results as can be seen if we replace the flawed query <DET> <N> with the more precise one:

<DET+s> <N+s> + <DET+p> <N+p>

This query reduces the size of the concordance from 22,142 to 8,517 entries!

... Elements of one's composition that are not to be eliminated...
... It's you that are out of your mind...
... didn't you leave all that behind you in Rome?...

(*are* is a noun that represents a metric unit of area). A syntactic parser could help NooJ decide that in these cases, *that* is a pronoun and thus remove these results.

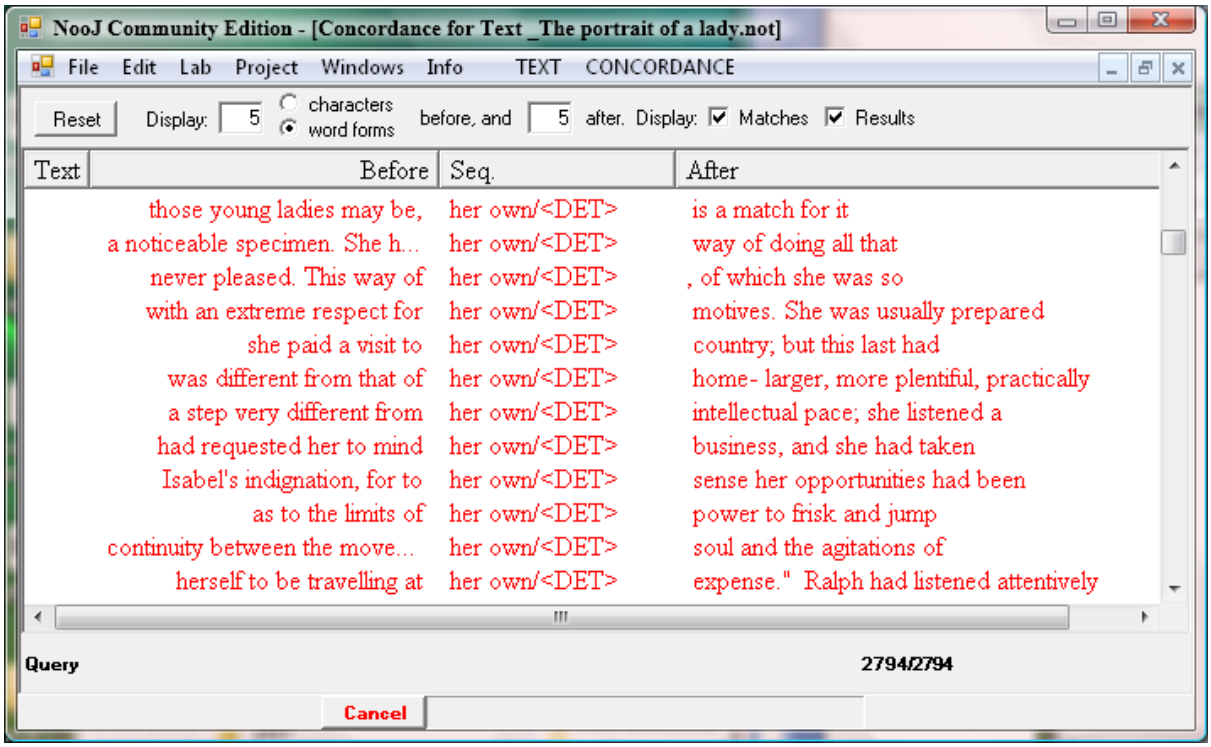
... that Baltimore was a Western city ...
... that Bantling hasn't ...
... we had left that behind long ago ...

However, in these cases, even if NooJ had performed a full syntactic analysis of these contexts, it would still have no way of rejecting these results: from a strictly linguistic point of view, *that* could still be a determiner and *behind* could be a noun...

In conclusion: there are some sequences such as “that appeals” that can be disambiguated with minimum effort: local grammars constitute a very efficient way to remove a large number of ambiguities and thus to clean up the result of users’ queries. We do not yet have a full syntactic parser that would help remove a number of ambiguities; however, even if we had a full syntactic parser, there would still be a number of residual ambiguities: it would thus be unavoidable to give users the possibility of removing ambiguities by themselves.

Automatic disambiguation

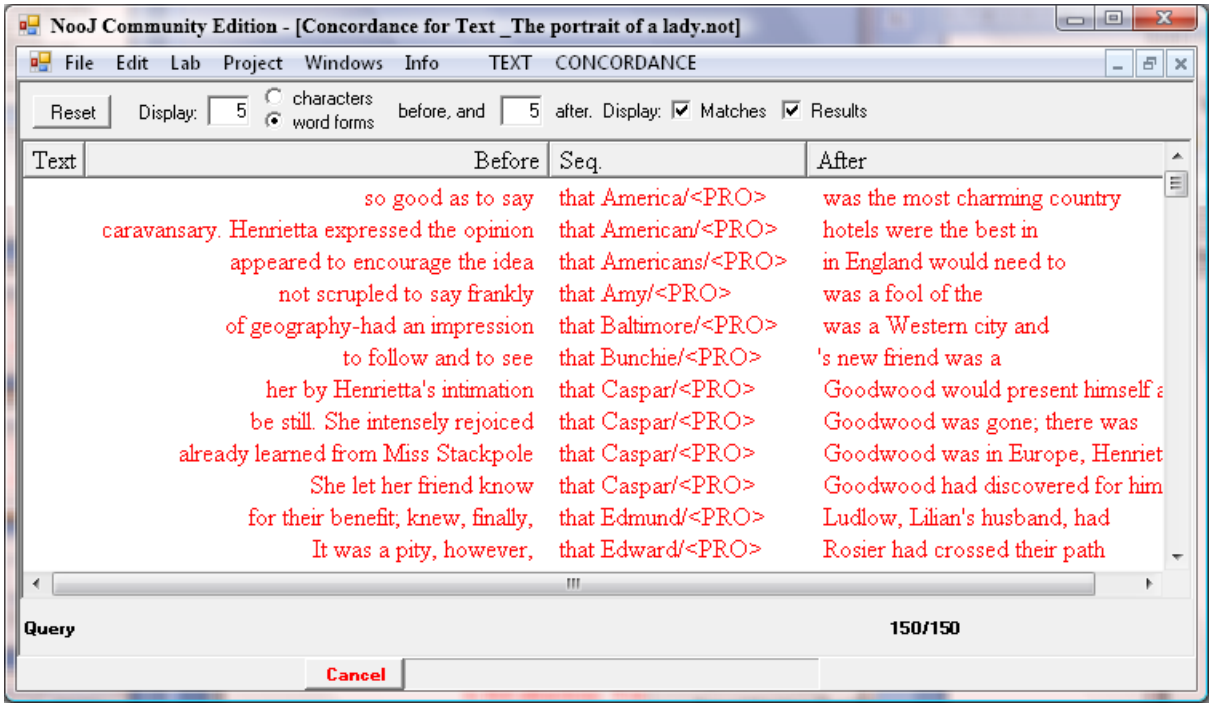
Local grammars can be used to remove ambiguities automatically. For instance, consider the local grammar in Appendix A that can be used to disambiguate half a dozen very frequent grammatical words which are systematically ambiguous (determiner or pronoun). The grammar merely lists a number of “unambiguous” contexts in which one can disambiguate these words definitively. For instance, *her* followed by *own* has to be a determiner as we can easily see in the concordance below. Although this local grammar is very specific, it covers 2.5% of the text. Our goal is to develop a set of 30+ local grammars such as this one: target frequent ambiguous words to be as efficient as possible, while at the same time as specific as possible, so as to avoid producing incorrect results.



Considering the context *her own* is enough to disambiguate *her* as a determiner

Semi-automatic disambiguation

The double constraint of efficiency and correctness is often very hard to follow, and in a number of cases, it is much too tempting to design very efficient rules which are correct in all but a few cases … For instance, it seems that all occurrences of the word *all* followed by a proper name correspond to the pronoun:

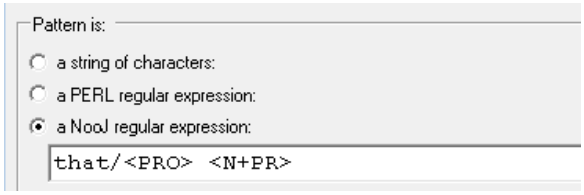


The context *that* <*N+PR*> is a good candidate for a disambiguation rule

This rule would have been very helpful to get rid of the incorrect concordance entries for the above-mentioned query <DET> <N>. Indeed, in the text *The portrait of a lady*, this rule can effectively be used to remove 150 ambiguities. However, it is not very difficult to build an example in which *that* is followed by a proper name, but is still a determiner:

Are you speaking of that John Doe?

Therefore, we need to be able to quickly enter simple disambiguation rules and check them before effectively applying them. In NooJ's new v2.2 version, it is possible to type in a query in the form of an expression associated with an output:



The regular expression **`that/<PRO> <N+PR>`** matches all the sequences of *that* followed by a proper name; then the output **`<PRO>`** is used to disambiguate the word form *that*.

We can edit the resulting concordance in order to filter out incorrect matches, i.e. sequences that were recognized but should not be used. We first select the unwanted concordance entries, then use the command **Filter out selected lines** of the CONCORDANCE menu to remove them from the concordance, and then Annotate Text (add/remove annotations) to perform the disambiguation for all remaining concordance entries.

This process is very quick: users should not hesitate to enter disambiguation commands that are very efficient but have poor accuracy, such as:

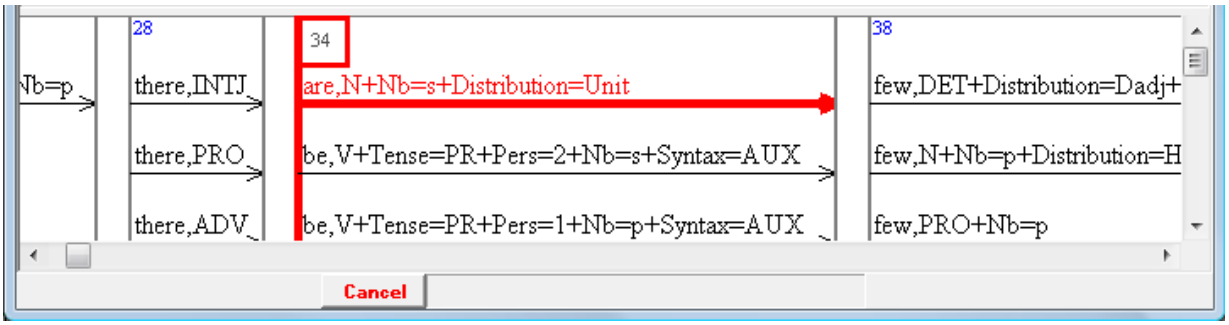
```
are/<V>
for/<PREP>
its/<DET>
```

etc.

The possibility of examining the resulting concordances and then filtering out incorrect analyses allows us to experiment more freely. Finally, note that we can save concordances at any moment: this allows us to validate large concordances at our own pace.

Editing the TAS

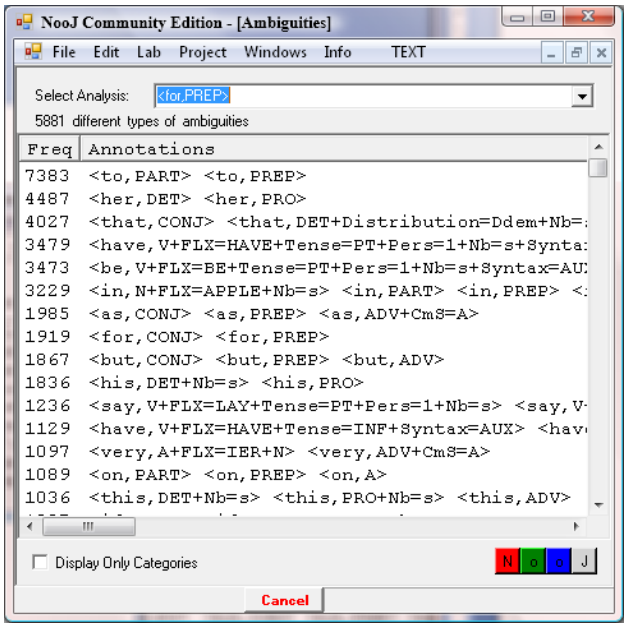
Sometimes, it is much faster to just delete unwanted annotations without having to write local grammars or even queries. In NooJ's new 2.2 version, we can just open the TAS (click **Show Text Annotation Structure** at the top of the text window), click an unwanted annotation, and then delete it either with the menu command **Edit > Cut**, or by pressing any of the following keys: Ctrl-x, Delete or Backspace. NooJ manages an unlimited number of undo's: **Edit > Undo** or press Ctrl-Z.



Examining the text's ambiguities

A new tool has been added to NooJ: the possibility of listing ambiguities according to their frequencies, to select one ambiguity and then to apply one solution to the text automatically. The most frequent ambiguous words will be the ones we intend to study in order to disambiguate the text efficiently.

Double-click **Ambiguities** in the text's result window, then click the header "**Freq**" to sort all ambiguities according to their frequency.



Here we notice that the word form *for* is ambiguous and occurs 1,919 times in the text. We select the second disambiguation solution (at the top of the window): <for,PREP>. Then, we click one of the colored buttons to build the corresponding concordance. Finally, we filter out the rare cases in which *for* is actually a conjunction (I did not find any occurrence in the text *Portrait of a lady*) and then click Annotate text (add/remove annotations) to disambiguate the word form.

Conversely, we can double-click Unambiguous Words in the text's result window in order to display the list of all unambiguous words. Usually, the word forms *the*, *a* and *of* appear most frequently. We can then use frequent unambiguous words as anchors, to remove word ambiguities that occur immediately before or after them. For instance, the word *table* is theoretically ambiguous (it could be a verb), but in the sequences *the table* and *a table* it has to be a noun. Focusing on frequent unambiguous words can help us design new types of disambiguation rules.

Perspectives

In its latest version, NooJ provides a number of tools to deal with all types of ambiguities that are stored in the Text Annotation Structure. Efficient local grammars can remove a large number of ambiguities automatically, and simple enhanced queries can be used to disambiguate a large number of occurrences of a very specific context for a grammatical word. Furthermore, the new ability to directly edit the TAS should help linguists clean up large texts. Moreover, NooJ's users can now display the most frequent ambiguities, as well as the most frequent unambiguous words, which should help them design efficient new local grammars rapidly.

We hope that these new tools will be shortly put to the test: we think that it is very reasonable to build a set of 30+ reliable local grammars that could get rid of 50% of ambiguities, and that semi-automatic and manual tools could be used to get rid of most "annoying" remaining ambiguities in order to build a new set of disambiguated corpora.

References

Silberztein Max, 2002-. *NooJ Manual*. Available for download at the WEB site http://www.nooj4nlp.net, 200 pages.

Silberztein Max, 2005. NooJ's Dictionaries. In the Proceedings of the *2nd Language and Technology Conference*, Poznan.

Silberztein Max, 2007. An Alternative Approach to Tagging. *Invited talk* In Proceedings of NLDB 2007. LNCS series, Springer-Verlag, pp. 1-11.

Silberztein Max, 2008. Frozen expressions and discontinuous annotations. In Proceedings of NooJ 2007, Barcelona. Cambridge Schooling Press.

Vietri Simona, 2008. The Formalization of Italian Lexicon-Grammar tables in a NooJ Pair Dictionary/Grammar. In Proceedings of NooJ 2008, Budapest. Cambridge Scholars Press.

Appendix A : a disambiguation grammar